Multimed Tools Appl (2009) 43:203–224
DOI 10.1007/s11042-009-0270-3

# SMIL State: an architecture and implementation for adaptive time-based web applications

**Jack Jansen · Dick C. A. Bulterman**

Published online: 8 April 2009
© Springer Science + Business Media, LLC 2009

**Abstract** In this paper we examine adaptive time-based web applications (or presentations). These are interactive presentations where time dictates which parts of the application are presented (providing the major structuring paradigm), and that require interactivity and other dynamic adaptation. We investigate the current technologies available to create such presentations and their shortcomings, and suggest a mechanism for addressing these shortcomings. This mechanism, *SMIL State*, can be used to add user-defined state to declarative time-based languages such as SMIL or SVG animation, thereby enabling the author to create control flows that are difficult to realize within the temporal containment model of the host languages. In addition, SMIL State can be used as a bridging mechanism between languages, enabling easy integration of external components into the web application. Finally, SMIL State enables richer expressions for content control. This paper defines SMIL State in terms of an introductory example, followed by a detailed specification of the State model. Next, the implementation of this model is discussed. We conclude with a set of potential use cases, including dynamic content adaptation and delayed insertion of custom content such as advertisements.

**Keywords** Declarative languages · SMIL · Multimedia web applications · Delayed ad viewing

## 1 Introduction

This paper examines technology to create adaptive time-based web applications. These are applications that use time as a major structuring paradigm, and need to adapt to changes at runtime. Such adaptation can be in the form of user interaction, but also other environmental changes such as location-based information or a change in available bandwidth. In addition to

J. Jansen (✉) · D. C. A. Bulterman
Centrum Wiskunde & Informatica (CWI), Science Park 123, 1098 XG, Amsterdam, The Netherlands
e-mail: Jack.Jansen@cwi.nl

D. C. A. Bulterman
e-mail: Dick.Bulterman@cwi.nl

D. C. A. Bulterman
VU University, De Boelelaan 1081,
1081 HV, Amsterdam, The Netherlands

being adaptive (or responsive), these applications should also be good web citizens: they (and the adaptation strategy) should be searchable, accessible, structured, reusable, etc.

Traditionally, the web has preferred structured declarative solutions over imperative ones: HTML [25], CSS [2], SMIL [4, 5], SVG [9] and many other web standards are all mainly declarative languages. The advantage of declarative languages in a web setting is that they facilitate reuse, accessibility and device independence [21]. However, at a lower level, imperative languages (mainly JavaScript [10]) are often required to enable time-dependent rendering, interactivity or binding of specific components. This presents a problem if we want to create adaptive time-based web applications, as these applications indeed require timing and interactivity and often the help of external components. The introduction of scripting into a webpage is a powerful tool, but therefore also a dangerous one: maintaining the advantages of the structured declarative model is not automatic, and may sometimes be impossible.

The alternative to structured declarative solutions is to use an imperative technology such as Flash [16]. Flash is an example of a proprietary binary format, which uses a content encoding that is—in its distribution format—difficult to parse at activation time. This forestalls search and (third-party) reuse. Moreover, any presentation and document adaptation and conditional accessibility need to be planned and explicitly catered for by the document author.

This is not to imply that declarative language already solve all interaction problems. If we examine the structured declarative languages that have an execution model (SMIL, SVG Animation), one piece of missing functionality is a user defined *data model*. A data model defines a document-specific collection of variables and settable parameters. Adding such a data model, while not completely eliminating the need for scripting, would allow a larger problem domain to be addressed without the need for a scripting language. This can make declarative documents more useful, especially in situations when document need to be generated automatically.

This paper introduces *SMIL State*, a technology that combines temporal web languages like SMIL or SVG with an external data model. SMIL State enables the use of free variables in declarative presentations, allowing the author to escape the temporal containment model in a controlled fashion. The data model is externalized, allowing it to be shared with other components and effectively enabling its use as an API between components of a web application.

This paper is an extended version of [18], which was presented at ACM Document Engineering 2008. It widens the scope of the former paper by examining how SMIL State is applicable to enriching existing SMIL content control mechanisms and by providing more detail on the implementation and the lessons learned from that implementation.

The paper is structured as follows. In section 2 we sketch the types of applications that are relevant for SMIL State, and describe an example of such an application in detail. We then outline the requirements of these applications. In section 3, we look at existing technologies for data model support, and investigate how well these match our requirements. Section 4 describes our SMIL State solution, as well as the motivations for our design. In section 5, we report on our initial implementation of SMIL State in the CWI Ambulant open source SMIL player. In section 6 we describe two example presentations and their architecture. We conclude with determining how well our solution matches our requirements, and some ideas about future work.

## 2 Scenario

In this paper we will concentrate on presentations which have time as their major structuring mechanism and that require user interaction/selection as the secondary

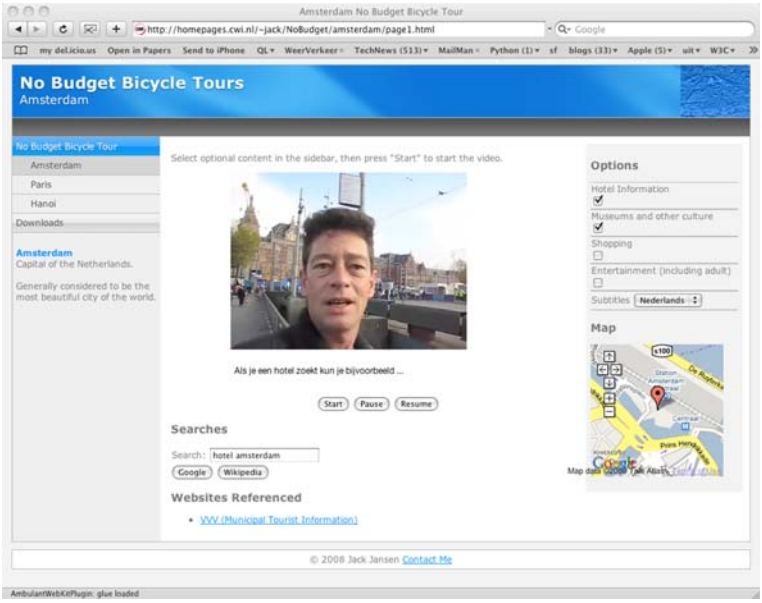


**Fig. 1** Screen shot of guided tour webapp

mechanism. To set the stage, let us start with an example of the type of presentations we want to enable.

The application shown in Fig. 1 is a web-based guided tour through Amsterdam.[1] The backbone of the application is a video, with the tour guide showing some highlights of the city, with additional information provided from a variety of external sources on hotels, shopping opportunities, entertainment and nightlife. The application allows viewers to select the topics in which they are interested dynamically: for example, if a viewer is staying with friends and prefers to be in bed right after dinner he can choose to skip the hotel and nightlife entertainment information. Of course, such choices are not static: the user should be able to change the content selection while viewing the presentation. If, in doing so, it turns out the cultural information is too detailed for his taste, he also should have the option to disable it on the fly.

The video presentation itself is rather fast-paced: the presenter races through the streets on his bicycle (as only a local can) and gives only terse information on the various subjects he encounters along the way. However, for each item he describes, the viewer is given the option of getting more information from external resources: when a museum is described, the link to the museum website is also given; the end user can temporarily pause the video to visit the museum website to find out about opening hours, etc. The presentation also includes a standard map, such as from Google Maps, orienting the viewing within the city. This has the benefit that the user may bookmark a place of interest, or again pause the presentation to search for related interesting places in the vicinity. The application also allows for the dynamic insertion of adwords, which bring up sponsored links relevant to the material currently presented. An interesting feature is that while sponsored links are triggered by location information, they are also temporally shifted so that their presentation

[1] A version of this example is available on the CWI Ambulant player website: http://ambulantplayer.org/smilStateExample.shtml.

is delayed until after a main content stream has completed. (This delayed scheduling can obviously be used for a host of applications beyond ad insertion.)

All these are examples of the use of *timed metadata* (*annotations*) in the presentation. The time logic of the presentation need only know which metadata pertains to which (timed) sections of the presentation. The actual presentation of the metadata is handed off to other components for rendering.

Our application example is similar in scope to the personalized multimedia tourist guide described by Scherp and Boll in [28], but where they generate personalized applications on the server, our solution allows *client-side personalization*. This not only distributes workload from the server to the clients, but has the added advantage that viewers can adapt their preferences during playback. Another form of adaptability that we aim for is *device independence*: depending on characteristics of the device on which the presentation is viewed (bandwidth, screen size), some content may be replaced by items more applicable to the current viewing context. If this could be done dynamically, so *session transfer* becomes possible, that would be an advantage: transferring the presentation to another device would then only require moving the presentation over to that other device as-is, the presentation itself would adapt to the new hardware characteristics.

Another important feature for presentation authors is *reusability*: if a general structure can be set up that handles multiple related presentations (such as bicycle tours for other cities, in our example), a significant authoring saving could be realized. It also eases the process of serving such presentations from a content management system. A related form of reuse is *third party enrichment*, which requires that it is possible to refer to portions of the presentation, either in-context or out-of-context. Such reuse is increasingly important on the web, and handled well for non-temporal media through wikis and blog syndication. We want to enable this form of reuse for multimedia presentations as well.

Finally, we feel *accessibility* is important. Not only does this enable the use of assistive technology, but it also allows search engines to index the content inside the presentation. This is another step in enabling third party reuse: to enable someone to refer to our content they must be able to find it first.

### 2.1 Requirements

To enable the type of applications sketched in the previous section we have a number of requirements on the technology we use. Let us outline our major requirements, so we can then determine how applicable various technologies are to our problem space.

The following requirements are important:

- The solution should be *structured*. Declarative structured languages have proven themselves to be facilitate reuse, accessibility, device independence and transformability.
- T*ime based* structuring is required, because time is a major structuring paradigm for the types of applications we envision. Having time as a first class citizen allows easier presentation creation and deep-linking. Time based structuring also enables close coupling of annotations with the media fragments they refer to, ensuring they stay together in the face of edits or deep linking.
- *Fragment support* on original media items is required. If there are multiple possible timelines through the presentation, lack of fragmenting original media would require the author to statically create multiple edits for each of the different timelines, or a large collection of small media snippets. Fragmenting support on the final presentation is also needed, again to enable third-party annotation.

- *Variables* are required to enable presentations to adapt to user input, especially if this adaptation is to happen at a different point in time than the input itself. Variables also enable interaction patterns not foreseen by the designers of the language.
- *Language bridging* is related to variables, but with a different scope. It is needed to enable integration of multiple components. Enabling multiple components allows the use of the best tool available for the sub-problem at hand. Language bridging and variables should also enable two-way communication between components, which increases the richness of the presentations possible.
- *Adaptability* is needed to enable platform independence, among other things. Built-in adaptability eases the burden on the author.
- *Accessibility* enables the use of assistive devices. Accessibility together with structuring enables search engines to index the content of the presentation.
- *Reusability* also eases the burden on authors, by allowing parts common to multiple presentations to be implemented only once. Content management systems and other dynamic methods of creating presentations benefit from it too, as only a single instance of common items needs to be stored. Third party modification and enrichment of existing presentation also requires reusability to be feasible without copying.

As will been seen, our SMIL State approach meets all of these requirements.

## 3 A Review of existing technology

Given the requirements of the previous section, this section will examine and evaluate the facilities available in existing Web technology. We will start with languages that aim at solving the whole problem space, or at least a large subset of it. Then we will look at emerging partial solutions that may be used to augment those solutions and other related work. We will then see how well all of these match our requirements.

### 3.1 Multimedia on the web

For interactive multimedia on the web there are currently two solutions in widespread use: Flash, and JavaScript combined with a plug-in to handle media playback (such as RealPlayer or, again, Flash). SMIL, which we will examine in greater depth in the following section, is not currently a serious contender in this market because it defines an execution model that is separated from the procedural control favored in web design.

The Flash solution is by far the most common, and used by websites like YouTube and Asterpix. All interaction is programmed explicitly in ActionScript [15], requiring specialized skills and tools. Moreover, due to the binary nature of the Flash distribution, the content is no longer easily accessible from outside. This is a problem for screen readers and other assistive technology, but also for web crawlers (content inside Flash does not show up in a search engine) and deep linking (no syndication or mash-ups).

Interactive multimedia presentations can also be created using standard technology: HTML, JavaScript and CSS. For audio or video playback this requires either the proposed HTML5 video extensions [13], or a plug-in to render the continuous media. While it is usually possible to control the media playback engine from JavaScript, for example starting and stopping video playback in response to user interaction, the reverse is usually not true: having the JavaScript react to events in the video (such as specific time codes) is not easy. In practice this means that using JavaScript is currently usually limited to presentations

using predominantly static media: if time is the primary structuring paradigm of the presentation Flash is a better solution. A prime example of doing multimedia presentations with only standard technology is the W3C Slidy tool [27], which can be used to create interactive accessible slideshows.

### 3.2 Declarative alternatives to scripting

Both technologies sketched so far share the property that the logic is expressed in a procedural language (JavaScript or ActionScript). If it were possible to express the logic in a declarative way that would be more suited to the trend in web languages towards declarative structuring to enable transformability, reuse and accessibility. An example of this trend is XForms [3], which uses a wholly declarative logic to specify not only the forms themselves but also the way these forms are connected to the underlying data store. In the context of this paper we are not so much interested in the model-view-controller paradigm of XForms or the high-level definition of the controls themselves (which allows an XForm form designed for a desktop web browser to be reused on a mobile browser, or even a voice browser [14]). We are, however, interested in the declarative nature in which constraints on input values can be specified, such as "weekday must be an integer between 0 and 6 inclusive". This feature means that old-style HTML forms that used procedural logic in JavaScript to check value constraints can be replaced by a declarative XForm.

XForms uses an XML document as its data model, and addresses the data items in this model through XPath expressions [7]. XForms 1.0 does not have an execution model, but it does not really need one for its application area. It does include a spreadsheet-like functional programming construct that allows variables to be computed on the basis of other variables, and that is good enough for its domain.

While it would probably be possible to create a complete interactive multimedia presentation using the technologies outlined in this section it would suffer from the fact that none of these languages have an inherent concept of time. Hence, all temporal relations would have to be explicitly coded in a language for which this was not the primary design goal.

### 3.3 SMIL

SMIL, the Synchronous Multimedia Integration Language is the W3C standard for presenting multimedia on the web. It is primarily an integration language: it contains references to media items, not the media data itself, and instructions on how those media items should be combined spatially and temporally.

SMIL is a declarative language: relations between media objects (and substructures) are described, and the computation of the timeline follows from this. The main temporal composition operators available are parallel composition, sequential composition and selection of optional content. Composition is hierarchical: nodes cannot become active unless all of their ancestors are active. The declarative containment model has one large advantage: SMIL presentations can adapt automatically to varying bandwidth conditions and alternate content with different durations because of the adaptive nature of hierarchical timing. The hierarchical temporal composition model is also a nice container for timed metadata, and allows structure-based deep linking into the content.

There are a number of mechanisms in SMIL that allow the presentation to react to user input (events) and to modify the behavior of other sections of the presentation (SMIL Animation) but

none of these break the basic containment model, they only modify behavior within those constraints.

The containment model has one serious drawback, though: there is no way in which the path taken through the presentation can be used to influence future behavior within that presentation. Or, more directly: there are no variables. In addition, with events being the only dynamic communication channel, a SMIL presentation can not exchange structured data with the outside world. This is a problem SMIL shares with many declarative languages. For example, functional languages have had to add constructs like effect classes [11] or monads [22, 30] to enable side-effects and input/output. Without these, their application domain would have been severely limited.

### 3.4 Other related work

The technologies described in sections 3.1 and 3.3 aim at addressing a large subsection of our problem space, but all have some shortcomings. In this section we will examine some ways to address those shortcomings and some solutions that address related problem areas, from which we may learn something.

XBL [12] is a language that allows an author to declaratively add behavior to an otherwise static HTML or XHTML document. It can modify the target document in-place, for example setting attributes on one element based on values obtained from another element. These actions can occur statically, somewhat similar to how XSLT [6] would operate on a document during load time, or dynamically, reacting to DOM events [26]. XBL has no notion of time or control flow, so using it to create self-paced multimedia presentations would be difficult.

XConnector [23] is an extension to XLink that has some overlap with XBL in application area. It also allows the specification of relations between different elements and attributes within an XML document. Some of these relations allow similar constructs as in XBL, such as changing an attribute value to match an attribute value elsewhere in the document. XConnector does have a notion of time, allowing the author to specify that something should start when something else stops, for example. The accompanying language XTemplate [24] allows an author to declare templates for such relationships, thereby enabling, among other things, the definition of temporal and spatial constraints on items in an HTML page in a way that facilitates reuse. XConnector and XTemplate together with HTML should enable creation of rich multimedia applications for the web fairly easily.

XHTML + SMIL [4] is similar to XConnector plus XTemplate, but more limited in scope: it allows the application of SMIL timing constructs to static HTML (or other XML) documents, thereby adding timing to an otherwise static format.

Another approach is taken by King, Schmitz and Thompson in [19] (unfortunately for reference purposes, no name is given for their work, so we will call it "*KST*" in this paper): adding rich transformations and expressions to a language that already has an execution model, such as SMIL or SVG animation. Where SMIL and SVG animation allow only a predefined number of operations on attribute values, determined by the language designers, this paper adds spreadsheet-like expressions and conditions through a functional "little language". The temporal constraints of SMIL animation are still in place, however.

Those temporal constraints are lifted by the same authors in [29], which adds a `<value>` element that can be used to store free variables. (It also adds a template mechanism, but that is outside the scope of this paper). This leads to a solution that has comparable application area and power as SMIL State within a single document, but the

externalized data model of SMIL State allows communication with the outside world, as well.

### 3.5 Comparison

Table 1 summarizes how existing technology matches the requirements from section 2.1. The first two columns show the main problems with the most popular current solutions: a finished presentation is a monolithic unstructured blob. This results in problems for deep-linking into a presentation, but also for accessibility, which also requires access to the internals of a presentation.

As we can see in the table, SMIL 2.1 does fairly well on the structuring front, but falls short in practical issues like rich interactivity and integration with other components (ignoring SMIL State, for the moment). Embedding XForms islands into a SMIL presentation does not help: it enables the end user to fill in forms that can be transmitted back to a server, but no extra interactivity is added. SMIL + XBL provides more options, but here the generality of what XBL allows would break some of the basic assumptions of SMIL, such as timegraph consistency. Incidentally, SMIL + JavaScript, which is not in the comparison table, would have the same problem.

KST is aimed at a different problem, but it still fits our requirements pretty well, with the exception of enabling communication with other components, which is outside its scope.

Interestingly enough, KST use different solutions in a number of areas where they were facing the same design decisions that our work considered:

- both solutions allow for rich data structures in the data model, but where we opted for XML for easy sharing, they felt a richer and more compact representation is needed;
- we think static strong typing is generally not needed for most applications, and can easily be added when needed through XSchema (following the model of XForms), their solution has static strong typing;
- their solution uses an expression language based on JavaScript expressions, ours uses XPath expressions, for standards compliance.

These different choices are partially dictated by different application areas, but probably partially by personal taste as well. We agree that XPath is not a very nice

**Table 1** Technology comparison

| | *F* | *JS* | *S2* | *S + X* | *XBL* | *KST* | *XCXT* | *XS* | *S3* |
|---|---|---|---|---|---|---|---|---|---|
| *Structured* | - | - | ✓ | ✓ | n/a | ✓ | +/− | +/− | ✓ |
| *Time based* | ✓ | - | ✓ | ✓ | - | ✓ | +/− | ✓ | ✓ |
| *Fragment support* | - | - | ✓ | ✓ | n/a | ✓ | ✓ | ✓ | ✓ |
| *Variables* | ✓ | ✓ | - | - | ✓ | ✓ | ✓ | - | ✓ |
| *Language bridging* | - | ✓ | - | - | ✓ | - | unknown | - | ✓ |
| *Adaptability* | +/− | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| *Accessibility* | - | +/− | ✓ | ✓ | n/a | ✓ | unknown | ✓ | ✓ |
| *Reusability* | - | - | +/− | +/− | +/− | +/− | ✓ | +/− | ✓ |

*F*: Flash; *JS*: JavaScript plus DOM access; *S2*: SMIL 2.1 (not including SMIL State); *S + X*: SMIL combined with XForms; *XCXT*: XConnector and XTemplate; *XS*: XHTML + SMIL; *S3*: SMIL 3.0 including SMIL State.

language to express complex expressions in, the corresponding expression in KST is definitely more readable. XPath expressions, however, are richer in the handling of complex data structures. In the case of static typing or not this is probably more a matter of personal preference.

XConnector and XTemplate are the best fit of the existing technologies, but it shares the XBL problem that they provide so much freedom that an author has to be careful not to lose the structuring advantages of the declarative model. The same is true for temporal structuring: this can be done by an author, but the language does not enforce it. We are also not sure whether XConnector provides any help with language bridging, the literature does not mention this.

XHTML + SMIL has similar advantages and shortcomings as SMIL 2.1, which is to be expected given their common heritage.

We will explain how SMIL plus SMIL State matches the requirements below.

## 4 Design and architecture

The main thrust of the research leading up to this article is that the addition of variables and communication would enable SMIL to be used in a number of application areas that are currently beyond its reach. These application areas include:

- *Courseware* is an important application area for multimedia software. One of the main advantages of using computers for instructional material is that the path through the material can adapt itself to the student. This takes the form of providing more in-depth material based on user interaction, either a "tell me more" button or the answer to a question being correct or not. Courseware also benefits from the ability to interact with problem domain specific components, to enable hands-on interaction or non-standard rendering capabilities. SMIL has no standard way to interact with external components, and no way to base decisions on user input that occurred earlier during the presentation.
- *Quizzes* are somewhat related, but here we also want to tally results, requiring computation. Moreover, quizzes are much more fun if your personal results can be compared to those of others, requiring communication of such dynamically computed scores to some central agent.
- *Games* are even more interactive, and require things like a ball to move in a direction determined by the mouse position when the ball hit a paddle, some time in the past. And a game needs more author-defined state, to determine when the aliens have all been destroyed. As with quizzes, destroying aliens becomes much more fun if your high score is transmitted to a server.

In addition, variables would allow an author to have more control over selectively rendered content. Prior to SMIL 3.0, SMIL provided *custom tests*, which allow end-user control over whether optional content is rendered or not, but the mechanism for presenting these options to the user is determined by the rendering user agent, not the author.

A separate, but related, issue with older SMIL releases is that it is impossible to communicate presentation state to the outside world. This problem becomes more acute once variables are added: if the SMIL presentation represents an interactive multiple-choice exam it is probably important to communicate the results to a server after the whole exam has been taken. If it represents a game we may want to keep high-scores at a central location.

A final design guideline was that the solution should be as simple as possible but be easily extensible if required for certain application areas.

### 4.1 SMIL State elements

SMIL State was designed using a two-tiered approach: first we architected hooks in the SMIL language to enable inclusion of a data model and expression language; we next focused on the selection of a default language for the data model and expression language. This layered approach has the advantage that if the default expression language is not the best choice for a given application it is possible to use another expression language that is more suitable without modifying the semantics on the SMIL level. The ability to use an expression language other than the default choice of XML and XPath, however, is not relevant to this paper, with the exception of the fact that it allows for extending the data model to the richer model supported by XForms.

The hooks in SMIL are:

- a `< state>` element in the head section of the document, used to declare the data model;
- an expr attribute that can be used on any timed element to conditionally skip the element;
- new timed elements `<setvalue>`, `<newvalue >` and `<delvalue>` which allow changing the data model;
- a head element `<submission>` and a timed element `<send>` that allow sending and receiving parts of the data model;
- an attribute value template construct, `{ expression}` , that can be used in selected attributes to interpolate data model values into attribute values;
- an event `stateChanged(ref)` that occurs when the specified item in the data model changes.

All of these hooks are modeled after existing SMIL constructs: `expr` behaves in a manner similar to system tests and custom tests, the timed elements behave like normal media items or SMIL animation elements. The attribute value template, which was modeled after the same construct in XSLT, fits in nicely with the existing mechanism in which SMIL animation and DOM access are allowed to modify attribute values in a running SMIL presentation (the so-called "sandwich model"). In this model, attribute value templates are only allowed in attributes where they cannot modify the time graph of a running presentation, similar to what is defined for SMIL animation.

For the default data model and expression language we have selected XML and XPath, respectively. We specifically allow XPath *nodeset* expressions: the data model is the XML document on which XPath operates, not the XPath variable bindings. XPath variables are used as the data model in some other standards such as DISelect [20], but this data model allows only simple unstructured scalar variables. Using the XML document as the data model allows structured values such as lists and associative arrays. To allow maintaining data model consistency, updates (by a single element) are atomic, and `<setvalue>` allows copying of subtrees.

The data model XML document may be embedded inside the SMIL document, but it is logically a separate document: the XPath expressions cannot refer to random items in the SMIL document.

**Listing 1**—Sample SMIL document with SMIL State constructs highlighted

```
<smil>
 <head>
  <state>
   <data xmlns="">
    <wantAd></wantAd>
   </data>
  </state>
 </head>
 <body>
  <seq>
   <par>
    <video src="match.mp4"/>
    <img xml:id="banner" begin="10s"
      end="15s" src="banner.png"/>
    <setvalue begin="banner.activateEvent"
      ref="wantAd" value="'commercial.mp4'"/>
   </par>
   <video expr="wantAd" src="{wantAd}"/>
  </seq>
 </body>
</smil>
```

Listing 1 shows an example of the use of SMIL State. The data model XML document is declared in the < state > element in the head section, it consists of a data root element with one child, wantAd, initially empty. The data and wantAd elements are not part of the SMIL language, this is really a separate XML document included inline for convenience only, hence the use of the xmlns attribute.

When the presentation starts, the match.mp4 video starts playing. After 10 s, the banner.png image is displayed for 5 s. If the user clicks on this image while it is active the value of the wantAd element in the data model is changed to the string commercial.mp4. The match.mp4 video continues playing until its end, whether or not the user clicks the image. After the video has finished the second video element get scheduled. Whether it plays or not depends on the wantAd data model item: if it is true (or non-empty and non-zero) it does play. Which video it plays depends on the value of the wantAd data model element, interpreted as a URL string.

### 4.2 Shared data model

The data model of SMIL State is external to the SMIL document itself. As stated in the previous section, this forestalls random changes to the SMIL document, thereby maintaining its time graph and its structural consistency. This has the effect that we do not lose the ability to do transformation and adaptation on the document, one of the key advantages of a declarative model.

The external data model has another advantage, however: it can be shared. In its simplest form this sharing can be between runs of the same presentation: an author can create a long-

running presentation that stores data when a section has been finished. A later run of the presentation can pick this up, and start the presentation at the given spot, in stead of at the beginning.

Sharing of the data model can also be applied to multiple components running at the same time. Using a shared data model as the communication paradigm between components decouples dependencies between these components: they only depend on a common understanding of the data model. This decoupling facilitates reuse, adaptability and retargeting: if a multimedia presentation wants to show locations on a map it only needs to define that it will store the location in `/location/latitude` and `/location/longitude`. The map applet can now listen for changes to these variables and modify the map view. Reuse is facilitated because another multimedia presentation only needs to be aware of this "*data model API*" to use mapping services. Same for adaptability and retargeting: if the map applet is replaced by a different one this does not affect the multimedia presentation. And even if the map applet is completely missing, for example because the presentation is viewed on a mobile device with not enough screen space to show both the presentation and the map, the multimedia presentation need not be aware of this.

### 4.3 Content control

SMIL has always supported optional content, the ability to render or skip content based on environmental conditions. *System tests* allow the presentation author to trigger on predefined conditions, such as available bandwidth and screen size, and *custom tests* allow extension of these with author-defined binary conditions. These constructs suffer from a number of drawbacks, however:

- there is no way to set the value of a custom test attribute from the presentation, and the user interface for defining these values is unspecified in the SMIL standard and left to individual implementations;
- the standard specifically allows system and custom tests to be evaluated once, at document load time, which limits their usefulness for interactivity and dynamic adaptation;
- the lack of an expression language means presentation authors can only test for attributes being true, not for them being false, and not for combinations of attribute values.

SMIL State integrates system tests and custom tests into its general expression framework. For example, the SMIL 2.1 construct

```
<audio src="background.ogg" systemBitrate="128000"/>
```

plays an audio fragment only if enough bandwidth is available. The corresponding SMIL State construct

```
<audio src="background.ogg" expr="smil-bitrate() &gt; 128000"/>
```

ensures dynamic evaluation, which means the presentation can adapt to varying bandwidth conditions (for example in mobile situations). Another example of the advantage of rich expressions is the ability to specify

```
<audio src="background.ogg" expr="smil-bitrate() &gt; 128000
and not(smil-audioDesc())"/>
```

This plays the background music track only if enough bandwidth is available, and if it does not interfere with audio descriptions.

## 5 Implementation

We have implemented SMIL State in our open source Ambulant SMIL player, this implementation was used to experiment with our sample applications. The implementation follows the two-tiered approach of the SMIL state design:

- architectural hooks into the SMIL language have been implemented in the core SMIL engine;
- data model and expression language are implemented in optional plug-in modules.

In this section we will look at three example expression language implementations: the *XPath-WebKit-state* module which was used for the guided tour, the *XPath-standalone-state* module used for the delayed advertisement presentation and a *Python-state* module.

**Listing 2** — State component API

```
class state_component_factory {
  state_component *new_state_component(const char *uri);
};

class state_component {
  void declare_state(const lib::node *state);

  void set_value(const char *var, const char *expr);
  bool bool_expression(const char *expr);

  void want_state_change(const char *ref,
       state_change_callback *cb);
    ...
};

class state_change_callback {
  void on_state_change(const char *ref);
};
```

The basics of the interface between the core interpreter and the expression language plug-in are shown in listing 2. Each plug-in provides a `state_component_factory` interface. The core iterates over these, passing the expression language specified by the document author as a parameter. A plug-in that implements this language will return its implementation, and the iterating stops.

Now the core calls `declare_state` passing the `<state>` element to initialize the data model. During runtime, methods such as `set_value` and `bool_expression` are called to implement the corresponding SMIL State elements and attributes. `StateChanged` events are implemented by the core calling `want_state_change` to signal its interest in changes to a specific data model item. The plug-in will now call `on_state_change` whenever the item is modified.

The advantage of using plug-in modules and a factory class for implementing the data model and expression language is that it enables multiple implementations. One use for

multiple implementations is the selection of the SMIL State expression language: the *Python-state* factory will return its implementation only if the SMIL author has specified that Python is to be used as the expression language. Another use of the factory class is that it allows plug-ins to dynamically determine whether they are applicable: both *XPath-WebKit-state* and *XPath-standalone-state* implement XPath as the expression language, but *XPath-WebKit-state* will only return its implementation after determining that the SMIL interpreter is currently hosted in a WebKit plug-in [1].

The *XPath-standalone-state* implementation is rather mundane: 700 lines of C++ that use the DOM and XPath facilities of the Gnome libxml2 [17] to implement SMIL State for standalone documents.

The *XPath-WebKit-state* implementation is more interesting: the application requires that it interfaces with a browser DOM and Javascript implementation as well as with an XForms implementation. Programming this directly in C or C++ using the NSAPI browser plug-in API would be painful: NSAPI is rather old, and its model is low level and verbose. As an example of how verbose it is, the following JavaScript statement obtains the base URL of the currently displayed HTML page:

```
base = document.location.href;
```

The equivalent C++ code is 50 lines long. Exporting functions from C++ to JavaScript also requires similarly verbose hand-written code.

Safari on MacOSX not only supports NSAPI-based plug-ins but also native *WebKit plug-ins*. These plug-ins are written in Objective-C, and tie in well with the AppKit and Foundation toolkits that are commonly used on OSX to create applications. Because Objective-C is a modern high-level language, it supports fairly rich introspection features, and the WebKit plug-in API exploits this to transparently bridge Objective-C to JavaScript and vice-versa: the plug-in can access JavaScript objects (and, hence, DOM objects) relatively easily, and exporting objects from Objective-C to JavaScript is similarly easy. Objective-C, in turn, is transparently bridged to Python through *PyObjC*, which is a standard component of MacOSX. And as the full Ambulant API is also transparently bridged to Python, through a modified version of the standard (but little known) Python tool *bgen,* the *XPath-WebKit-state* implementation is now a mere 200 lines of Python.

Figure 2 shows the cascade of bridges mentioned in the previous paragraph, and despite their rickety appearance we have not experienced stability issues. The use of dynamic languages and the availability of the language bridges has enabled us to use rapid prototyping methods to perform these experiments. The SMIL State design matched the platform nicely, and clean separation of components was almost automatic. The only link between the WebKit world and the Ambulant world is DOM access and XML Events, between the WebKit DOM and the Ambulant SMIL State plug-in. The relevant components in this implementation are shown in Fig. 3.

We have also started thinking about implementing browser integration through the standard NSAPI plug-in API, to facilitate using SMIL State in Firefox or Internet Explorer. Experience with the WebKit plug-in suggests that providing a general bridge to a high-level

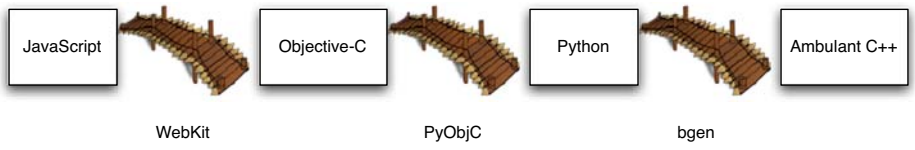


**Fig. 2** Implementation language bridging

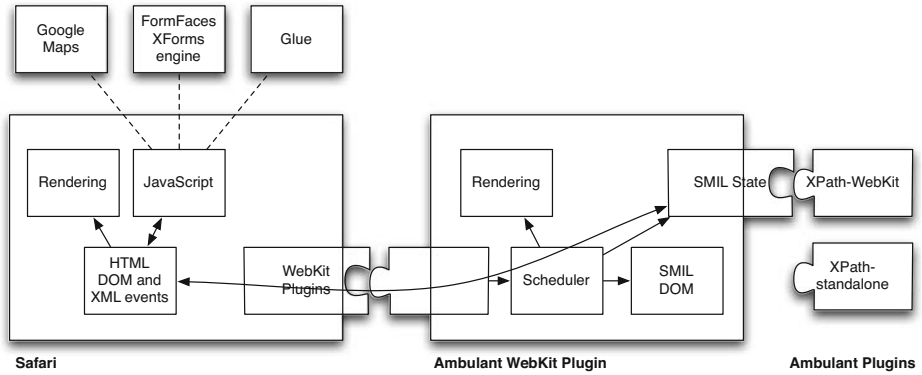


**Fig. 3** Browser plug-in implementation

language may be a good solution that allows easy experimentation, so we are looking at implementing a generalized JavaScript-Python bridge based on NSAPI.

## 6 Applications

In this section we examine two applications that address the two different aspects of using SMIL state. We start with a full-blown web app as outlined in section 2 and continue with a much more lightweight presentation that enables ad insertion into video presentation without the end-user annoyance that it currently often evokes.

These applications were created using our Ambulant SMIL playback engine, with support for SMIL State added. In case of the first application Ambulant was hosted in the Safari web browser, together with the FormFaces XForms implementation and the Google map applet. The second presentation runs in a standalone Ambulant player.

### 6.1 Guided tour webapp

We now revisit the example presentation sketched in section 2, and show how it was designed.

The general control flow of the application is driven by SMIL, and consists of a linear sequence of video clips, with optional subtitles. Some clips, such as the introduction, are played unconditionally, others are played or skipped depending on user preferences set through the XForms controls. For each clip, the lattitude and longitude information are stored in the data model. The location is picked up by glue on the webpage and communicated to the map applet. Additionally, references to relevant external websites, adwords and search terms are put in the data model. This information is picked up by glue code in the webpage and displayed.

Because multiple components are involved (SMIL for media playback and timing control, XForms for interaction, map applet) HTML is used as the outermost container format, as well as for displaying additional content such as background links, etc. The global structure of the presentation is shown in Fig. 4: the HTML document embeds the XForms form and the applet, and it has a reference to the SMIL presentation. SMIL (through SMIL State) and XForms both refer to the shared data model, and can both read and modify it. The map applet and HTML page itself only read values from the data model, through a bit of glue. How this architecture matches to the visual representation on the web page is shown in Fig. 5.

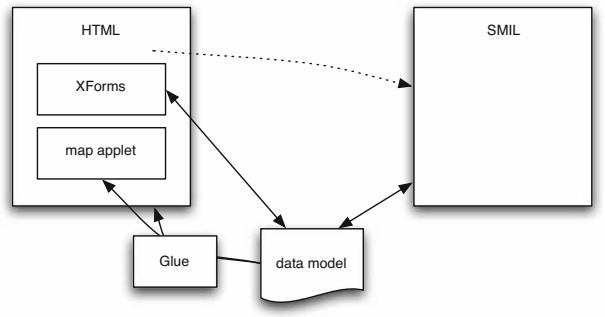


**Fig. 4** Guided tour document model

The glue needs a bit more explanation: as only XForms and SMIL have direct access to the data model, in the prototype the glue is implemented with a bit of Javascript, triggered by DOM events when the data model changes. This glue could be implemented using XBL, XConnector or another declarative form, but unfortunately none of these were available in a browser that could also host our SMIL plug-in.

Listings 3 and 4 show the relevant parts of the HTML and SMIL documents, respectively. The HTML document has the embedded data model (in the XForms namespace). It consists of sections `optionalContent`, for content selection, `subtitles`, for subtitle selection, and `gps`, `backgroundLinks`, `backgroundSearch` and `adWords`, for communicating timed metadata. The XForms form enables the viewer to select, for example, whether to display the hotel information or not.

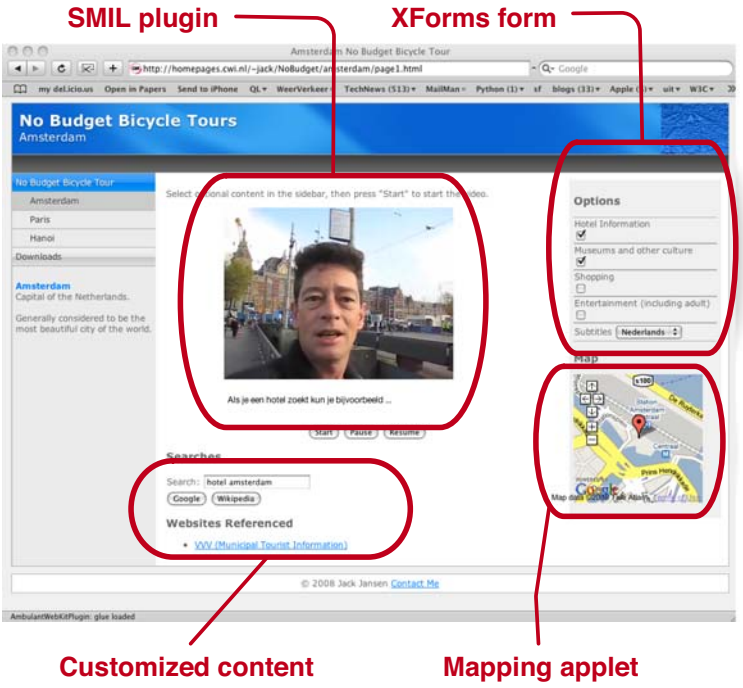


**Fig. 5** Mapping of components to screen rendition

In the SMIL code, the whole section is played only if `optionalContent/hotels` is true. The multimedia data for that section consists of a subsection of the video clip and some subtitles. The metadata is stored in the data model at the time the media start. Some of this metadata is scalar (such as longitude, latitude and adWords), some is structured (background search items). In the latter case a new sub-item named `hotel` is added to the `backgroundSearch` container.

**Listing 3**—HTML and XForms code

```
<head>
 <form:model ...>
  <form:instance id="jacksinstance">
   <data xmlns="">
    <optionalContent>
     <hotels>false</hotels>
     <culture>true</culture>
     <shopping>false</shopping>
     <entertainment>false</entertainment>
    </optionalContent>
    <subtitles>none</subtitles>
    <gps>
      <long></long>
      <lat></lat>
    </gps>
    <backgroundLinks/>
    <backgroundSearch/>
    <adWords/>
   </data>
  </form:instance>
  ...
 </form:model>
 ...
</head>
<body>
 ...
 <form:select ref="optionalContent/hotels" ...>
  <form:label>Hotel Information
  </form:label>
  <form:item>
   <form:label></form:label>
   <form:value>true</form:value>
  </form:item>
 </form:select>
 ...
</body>
```

Note that, despite the similarity to SMIL Animation constructs like `<set>` , these `<setvalue>` and `<newvalue>` elements are not automatically reverted when their timeline ends. In that way, they form the procedural escape hatch for the temporal containment model, while still keeping that containment model intact in the general case.

**Listing 4**—SMIL code

```
<par expr="optionalContent/hotels">
 <video src="biketour.mp4"
   clipBegin="26s" clipEnd="53s" .../>
 <smilText expr="subtitles = 'nl'" ...>
   Als je een hotel zoekt kun je
   bijvoorbeeld ...
   <clear begin="6s"/>
   ...
 </smilText>
 <setvalue ref="gps/long" value="52.3776"/>
 <setvalue ref="gps/lat" value="4.89868"/>
 <setvalue ref="adWords" value="'hotel amsterdam'"/>
 <newvalue ref="backgroundSearch"
     name="hotel" value="'hotel amsterdam'"/>

 ...
</par>
```

### 6.2 Delayed ad selection

The standard way to do advertisements in video streams, whether over the internet or through traditional channels, is ad insertion. This can be static or dynamic, but the dynamism is generally server-based: depending on data the server knows it selects specific ads to insert. This selection process may be based on a user profile the server keeps, but there is no direct user interaction. Ad insertion done dynamically at client side, based on user interaction, such as discussed in [8], has a different problem: it hinges on the fact that the viewer is so interested in the product that she actually clicks the link, disrupting her viewing experience. We feel this may be an unlikely general model.

For static media on the internet the situation is wholly different. Inserted advertisements, which require the user to first read the ad before being able to get at the target content, are generally frowned upon, and recently most major browsers contain features that actively try to forestall pop-ups and other disruptive advertisements. Instead of the forced consumption of ads, web pages tend to work with the voluntary model: the user has the option of clicking a banner ad. While even this may go too far for some people, the model probably will have a much larger acceptance than forced ads.

We feel that it would be good to transport the voluntary banner ad method to the realm of multimedia. However, if the user is in the mindset of watching a video it may be unlikely that this user clicks the advertisement instead.

To address this issue, we have come up with a technique we call *delayed ad viewing*. A video program has pre-determined advertisement slots, and during such a slot an advertisement always plays. However, through interaction with the presentation before the advertisement slot the user can influence which ads will be played.

The sample presentation consists of a (non-live) football program. Included in the presentation are a number of commercial videos, with a default playout order. At various times, usually when a billboard is in plain view in the video footage, a banner for a specific brand will show up in the lower-right corner of the screen for a couple of seconds. Figure 6 shows how this looks during playback. If the user clicks during this period the


**Fig. 6** Video with banner for delayed advertisement showing in bottom right corner

corresponding ad will be moved to the front of the playout list. When it is time for a commercial break, the main video is paused and the head of the advertisement playout list is shown. After an advertisement has been viewed its banner will no longer show.

Figure 7 shows the timelines of three different playbacks of the same document. User 1 did nothing and got the default ad playout order of a soap advertisement and a beer advertisement. User 2 clicked the "Ford" banner, and got that advertisement first followed by the default soap ad. User 3 requested the Amstel and Ford banners, and was spared the Lux ad. (At least, during the first commercial break!)

At the end of the presentation the state variables contain information on which ads have been watched. This information could be transmitted back to a central server for monetization, along the lines of pay-per-click ads on static web pages. Alternatively, this data could be gathered by the media server when the request to serve the ad stream comes in.

Note that the use of delayed ad selection does not preclude other current standard ad-insertion methods. The SMIL presentation can be generated on demand by the server for a specific user. Whether the user has complete freedom to select advertisements or only limited options is a choice at the discretion of the content provider. Different advertisement selections, choices and commercial break frequencies can be served to different users by serving only different SMIL documents: the underlying media items can all be static.

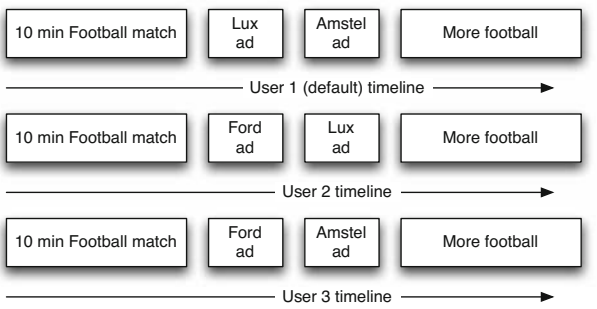


**Fig. 7** Different playout orders

The structure of the presentation is rather simple, and listing 1 gives the general idea. A problem that was encountered is that the XPath expression language is primarily meant for manipulating general XML documents and not for the more spreadsheet-like operations we are using it for. Hence, functions like `max()`, which would have made the ad reordering a lot simpler, are missing and the logic needs to be written out.

## 7 Conclusions and future work

Based on the example applications we have created we can match our solution to the requirements (see Table 1). SMIL State does not interfere with any of the advantages of SMIL, so we only need to look at the three requirements where SMIL is lacking:

- *Variable support* works nicely in SMIL State, and simple use cases have simple solutions. XPath as the expression language could have used a little boost, though, as XForms did by introducing a number of convenience functions into the XPath function namespace. Even so: XPath may be a rich language to encode expressions, it is not a very user-friendly one. We plan to investigate whether it is possible to come up with an alternative for XPath that is as easy to use as KST or JavaScript while still allowing the use of a full XML document as the underlying data store.
- *Language bridging* works fine. Here the problem is on the other side of the bridge: as only SMIL State and XForms currently share this data model, the integration into other languages requires some glue code.
- *Reusability* works fine. Whether you want to replace components or refer to fragments inside the presentation, we have not encountered any problems.

SMIL State has been proposed to the SYMM working group, and has been accepted as a part of the standard for SMIL 3.0, which became a W3C Recommendation in December 2008.

The model is fairly easy to support, as is demonstrated by our multiple implementations, as well as by an independent third-party implementation which was required for inclusion in the SMIL 3.0 Recommendation .

We intend to pursue and extend this model in the context of the W3C Rich Web Application Backplane Incubator Group. There we will also try and address the some of the shortcomings sketched in this section: Current web application toolkits are (naturally) oriented towards procedural languages, specifically JavaScript. A more declarative interface, possibly based on using XBL to connect widget-like components would not only benefit our model, but also help accessibility and general reuse.

We also plan to experiment with more rich interaction with the environment, through the *Python-state* implementation. This implementation should allow things like controlling and interrogating external applications, which could be put to good use for “hands-on” style courseware and such.

*Distributed shared state* is another area that has our interest: with the ubiquitous availability of handheld devices that have decent connectivity, compute power and rendering capabilities it is interesting to look at the possibility to create presentations that are shared among devices in a loosely coupled manner.

**Acknowledgements** The work reported in this paper has benefited from suggestions offered by members of the W3C backplane activity and members of the W3C Synchronized Multimedia working group. Sjoerd Mullender, Julien Quint and Daniel Weck have provided comments on earlier versions of this research. We are grateful to Steven Pemberton for introducing us to the philosophy behind XForms, which seeded the

design of our solution. This work has been funded by the NWO BRICKS PDC3 project, and by the FP7 IST project TA2. Development of the open source Ambulant Player and CWI's participation in the SMIL standardization effort have been funded by the NLnet foundation. We gratefully acknowledge this support.

## References

1. Apple Inc (2008) WebKit Plug-In Programming Topics. Available at: http://developer.apple.com/documentation/InternetWeb/Conceptual/WebKit_PluginProgTopic/WebKit_PluginProgTopic.pdf
2. Bos B, Lie H, Lilley C, Jacobs I (1998) Cascading Style Sheets, level 2. Available at: http://www.w3.org/TR/CSS2/
3. Boyer J (2007) XForms 1.0 (Third Edition). W3C. Available on: http://www.w3.org/TR/xforms/
4. Bulterman D, Rutledge L (2008) SMIL 3.0: Interactive multimedia for the web, mobile devices and daisy talking books. Springer-Verlag, Heidelberg, Germany. ISBN 978-3-540-78546-0
5. Bulterman D et al (2008) Synchronized Multimedia Integration Language (SMIL 3.0). W3C. Available on: http://www.w3.org/TR/SMIL/
6. Clark J (1999) XSL Transformations (XSLT) Version 1.0. Available at: http://www.w3.org/TR/xslt
7. Clark J, DeRose S (1999) XML Path Language (XPath) Version 1.0. Available at: http://www.w3.org/TR/xpath
8. Costa R, Moreno MF, Rodrigues RF et al (2006) Live editing of hypermedia documents. DocEng '06: Proceedings of the 2006 ACM symposium on document engineering. ACM, New York, NY, pp. 165–172. doi: 10.1145/1166160.1166202
9. Ferraiolo J, Fujisawa J, Jackson D et al (2003) Scalable Vector Graphics (SVG) 1.1 Specification. Available at: http://www.w3.org/TR/SVG11/
10. Flanagan D (2006) Javascript: the definitive guide. O'Reilly & Associates, Sebastopol, CA, USA. ISBN 0-596-10199-6
11. Gifford D, Lucassen J (1986) Integrating functional and imperative programming. ACM conference on LISP and functional programming. doi: 10.1145/319838.319848
12. Hickson I (2007) XML Binding Language (XBL) 2.0. W3C. Available on: http://www.w3.org/TR/xbl/
13. Hickson I et al (2009) HTML 5 Draft Recommendation. Available at: http://www.whatwg.org/specs/web-apps/current-work/. Retrieved on February 2, 2009.
14. Honkala M, Pohja M (2006) Multimodal interaction with xforms. ICWE '06: Proceedings of the 6th international conference on Web engineering. ACM, New York, NY, pp. 201–208. doi: 10.1145/1145581.1145624
15. http://www.adobe.com/devnet/actionscript/
16. http://www.macromedia.com/software/flash/about/
17. http://xmlsoft.org/
18. Jansen J, Bulterman D (2008) Enabling adaptive time-based web applications with SMIL state. DocEng '08: Proceedings of the 2008 ACM symposium on Document Engineering (2008). ACM, New York, NY, USA. doi: 10.1145/1410140.1410146
19. King P, Schmitz P, Thompson S (2004) Behavioral reactivity and real time programming in XML: functional programming meets SMIL animation. DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering (2004). doi: 10.1145/1030397.1030411
20. Lewis R et al (2007) Content Selection for Device Independence (DISelect) 1.0. W3C. Available on: http://www.w3.org/TR/cselection/
21. Lie H, Saarela J (1999) Multipurpose Web publishing using HTML, XML, and CSS. Communications of the ACM, Vol. 42, Issue 10. ACM, New York, NY, pp. 95–101. doi: 10.1145/317665.317681
22. Moggi E (1988) Computational Lambda-calculus and monads. In proceedings 4th Annual Symposium on Logic in Computer Science. IEEE Computer Society Press, Washington, DC
23. Muchaluat-Saade D, Rodrigues R, Soares L (2002) XConnector: extending XLink to provide multimedia synchronization. Proceedings of the 2002 ACM symposium on Document Engineering. ACM, New York, NY, USA. doi: 10.1145/585058.585069
24. Muchaluat-Saade D, Soares L (2003) XConnector and XTemplate: improving the expressiveness and reuse in web authoring languages. The new review of hypermedia and multimedia. Taylor&Francis, Bristol, PA, USA. doi: 10.1080/13614560208914739
25. Pemberton S et al (2002) XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). Available at: http://www.w3.org/TR/xhtml1
26. Pixley T (2000) Document Object Model (DOM) Level 2 Events Specification Version 1.0. Available at: http://www.w3.org/TR/DOM-Level-2-Events/

27. Raggett D (2006) Slidy-a web based alternative to Microsoft PowerPoint. XTech (Amsterdam, May 16–19 2006). Available on: http://www.w3.org/2006/05/Slidy-XTech/slidy-xtech06-dsr.pdf
28. Scherp A, Boll S (2004) Generic support for personalized mobile multimedia tourist applications. MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia (2004). doi: 10.1145/1027527.1027566
29. Thompson S, King P, Schmitz P (2007) Declarative extensions of XML languages. DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering (2007). doi: 10.1145/1284420.1284442
30. Wadler P (1990) Comprehending Monads. In Proceedings of the 1990 ACM conference on lisp and functional programming, pages 61–77, Nice, France, 1990

**Jack Jansen** Is a researcher at Centrum Wiskunde en Informatica (CWI), with over 25 years of experience in multimedia and distributed systems. Empowering people to put available technology to a use they themselves envision is his driving principle. This results in activities ranging from languages, such as Python, via web standardization work (SMIL, Rich Web Application Backplane) to implementing systems for accessible and reusable multimedia (Ambulant). Recently, he has finally started to pursue a PhD.

**Dick Bulterman** Is head of distributed multimedia systems research at CWI, the Dutch national center for mathematics and computer science in Amsterdam. He is also a professor of computer science at the VU University in Amsterdam. Dr. Bulterman received his Ph.D. in computer science from Brown University in Providence RI (USA) in 1981. He has been co-chair of the W3C working group on synchronized multimedia since 2007; this group released the SMIL 3.0 Recommendation in late 2008.

Bulterman has been active in the Document Engineering community since 2005. He is past program chair and past general chair of the ACM DocEng Symposium. He is also past chair of ACM Multimedia of and IEEE ISM.

Dick Bulterman lives in Amsterdam with his wife and two children.